

# Supplementary Data 2: Simulations of scRNA-seq data to analyse the effect of differentially expressed genes on normalization

*Catalina Vallejos, Davide Rissò, Antonio Scialdone, Sandrine Dudoit and John Marioni*

## Contents

|  |           |
|--|-----------|
| <b>Introduction</b>                            | <b>2</b>  |
| Simulation setup . . . . .                     | 2         |
| <b>Define functions and load packages</b>      | <b>2</b>  |
| <b>Estimation of parameters from real data</b> | <b>5</b>  |
| <b>Simulations</b>                             | <b>8</b>  |
| Symmetric differential expression . . . . .    | 8         |
| Asymmetric differential expression . . . . .   | 9         |
| <b>Results</b>                                 | <b>11</b> |

# Introduction

This document is provided as supplementary material for the article:

Normalizing single-cell RNA sequencing data: challenges and opportunities (2016). Catalina Vallejos, Davide Risso, Antonio Scialdone, Sandrine Dudoit and John Marioni

By using simulated data, here we explore how the presence of differentially expressed genes between groups of cells affects the performance of different normalization methods in single-cell RNA-seq (scRNA-seq) datasets. Alongside the most widely used normalization techniques developed for bulk RNA-seq data (Reads Per Million, RPM; DESeq; Trimmed Mean of M-values, TMM; Upper-Quartile, UQ), we estimate the scaling factors with a recently published normalization method specifically designed for scRNA-seq data (Scran). The simulation settings are chosen to mimic the features (e.g., overdispersion due to technical noise) of scRNA-seq data, with parameters estimated from a recently published dataset.

## Simulation setup

Throughout, we generate read counts  $X_{ij}$  from a negative binomial. We simulate read counts from two homogeneous groups of cells and set the negative binomial parameters equal to  $s_j\mu_iFC_i$  (mean) and  $\phi_i$  (over-dispersion), where:

- $s_j$  is a global scaling factor for cell  $j$ ,
- $\mu_i$  represents the overall expression rate for gene  $i$  across all cells
- $FC_i$  represents a fold change in expression between the first and the second group of cells
- $\phi_i$  is the overdispersion of gene  $i$ .

In order to simulate a realistic scenario, the scaling factors  $s_j$ , the expression rates  $\mu_i$  and the over-dispersions  $\phi_i$  are estimated from a real scRNA-seq dataset (Klein et al, Cell, 21:161, 2015).

We simulate two groups of 100 cells with a total of 10,000 genes. 1,000 genes are differentially expressed (DE) between the two groups with a fold-change equal to 5. Four different scenarios are analysed, corresponding to different proportions of up-regulated genes in the two groups of cells, as summarised in the table below:

| Setting               | DE genes   |
|-----------------------|--|
| Symmetric DE          | 50%/50% genes upregulated in the first/second group of cells |
| Asymmetric DE (60/40) | 60%/40% genes upregulated in the first/second group of cells |
| Asymmetric DE (80/20) | 80%/20% genes upregulated in the first/second group of cells |
| Fully asymmetric DE   | 100% genes upregulated in the first group of cells           |

For each scenario we run 100 simulations and compare the scaling factors estimated by the above listed normalization methods with the true ones.

## Define functions and load packages

```
#Results folder #####
results.path = "./"

#Packages #####
require(scran)
require(DESeq2)
require(edgeR)
```

```

#Functions #####
#RPM scaling factor
rpm <- function(x) {
  s <- colSums(x)
  n.cells<-ncol(x)
  s <- n.cells*(s/sum(s))
  counts <- t(t(x)/s)
  return(list(s=s, counts=counts))
}

#deseq normalization
deseq <- function(x) {
  s <- estimateSizeFactorsForMatrix(x)
  n.cells<-ncol(x)
  s <- n.cells*(s/sum(s))
  counts <- t(t(x)/s)
  return(list(s=s, counts=counts))
}

#tmm normalization
tmm <- function(x) {
  s <- calcNormFactors(x)*colSums(x)
  n.cells<-ncol(x)
  s <- n.cells*(s/sum(s))
  counts <- t(t(x)/s)
  return(list(s=s, counts=counts))
}

#upper quartile normalization
uq <- function(x) {
  s <- apply(x, 2, quantile, .75)
  n.cells<-ncol(x)
  s <- n.cells*(s/sum(s))
  counts <- t(t(x)/s)
  return(list(s=s, counts=counts))
}

#scran
scran_norm <- function(x) {

  clust<-quickCluster(x,min.size=50)
  s <- computeSumFactors(x, clusters=clust,sizes = c(20, 30,40,50))
  n.cells<-ncol(x)
  s <- n.cells*(s/sum(s))
  counts <- t(t(x)/s)
  return(list(s=s, counts=counts))
}

#run simulation

```

```

RunSim <- function(seed, mu, s, phi = 0, groups="no")
{
  # Simulation
  set.seed(seed)
  if(groups=="no"){
    if(length(phi) == 1)
    {
      if(phi == 0) { x = simPois(mu, s)}
      else { x = simNB(mu, s, phi) }
    }
    else
    {
      if(length(phi)!= length(mu)) stop("Length of 'phi' does not match the length of 'mu'")
      else { x = simNB_phi(mu, s, phi)}
    }
  }

  else if(groups=="yes"){

    x1 = simNB_phi((mu[[1]]), (s[[1]]), (phi[[1]]))
    x2 = simNB_phi((mu[[2]]), (s[[2]]), (phi[[2]]))

    x = cbind(x1, x2)

  }

  # Normalization
  x_rpm <- rpm(x)
  x_deseq <- deseq(x)
  x_tmm <- tmm(x)
  x_uq <- uq(x)
  x_scran <- scran_norm(x)

  list(s_rpm = x_rpm$s, s_deseq = x_deseq$s,
       s_tmm = x_tmm$s, s_uq=x_uq$s, s_scran = x_scran$s)
}

#negative binomial
simNB_phi <- function(mu, s, phi) {
  q <- length(mu)
  n <- length(s)
  return(t(sapply(1:q, FUN = function(i) { simNB(mu[i], s, phi[i])})))
}
simNB <- function(mu, s, phi) {
  q <- length(mu)
  n <- length(s)
}

```

```

    return(matrix(rnbinom(n*q, mu = mu %*% t(s), size = 1/phi), nrow=q, ncol=n))
}

#compute average ratio between simulated and true scaling factors in the two groups of cells
compute.ratio<-function(Sim, true.size){

  avg<-apply(Sim, 1, function(x){
    m<-do.call(rbind, x)

    return(colMeans(m))

  })
  ratio<-apply(avg,2, function(x) x/true.size)

  ratio.group1<-ratio[1:100,]
  ratio.group2<-ratio[101:200,]
  colnames(ratio.group1)<-paste0(colnames(ratio), ".group1")
  colnames(ratio.group2)<-paste0(colnames(ratio), ".group2")

  ratio<-cbind(ratio.group1, ratio.group2)
  ratio<-ratio[,c("s_rpm.group1", "s_rpm.group2",
                "s_deseq.group1", "s_deseq.group2",
                "s_tmm.group1", "s_tmm.group2",
                "s_uq.group1", "s_uq.group2",
                "s_scran.group1", "s_scran.group2")]

  return(ratio)
}

```

## Estimation of parameters from real data

We estimate the scaling factors, the mean expression values and the overdispersion of the genes from the scRNA-seq dataset published in Klein et al, Cell (2015). In particular, we use the mouse embryonic stem cells before LIF withdrawal and estimate the parameters for the top 10,000 genes after RPM normalization.

The mean-overdispersion relationship is fitted with a loess curve; this fit will be used to estimate the overdispersion of differentially expressed genes.

```

# counts <- read.csv("GSM1599494_ES_d0_main.csv.bz2", header=FALSE,
#stringsAsFactors=FALSE, row.names=1)
# colnames(counts) <- paste0("d0.", seq_len(ncol(counts)))
#save(list="counts", file="..../datasets/Klein2015/klein.RData")
load("../..../datasets/Klein2015/klein.RData")

NormRpm <- rpm(counts)#RPM normalization
Mu = rowMeans(NormRpm$counts)#Average expression

g<-names(sort(Mu, decreasing=T)[1:1e4])
counts<-counts[g,]#Select top 1e4 genes
NormRpm$counts<-NormRpm$counts[g,]

```

```

row.names(counts)<-paste0("gene.",1:1e4 )
row.names(NormRpm$counts)<-paste0("gene.",1:1e4 )

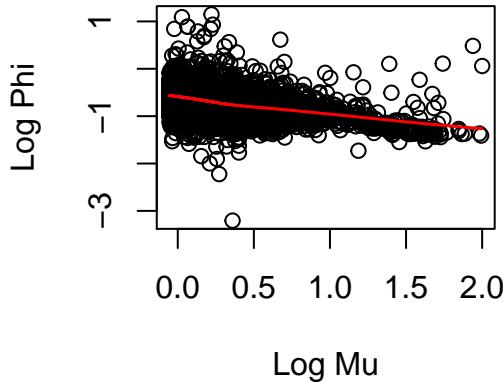
#Compute parameters for simulations
S = NormRpm$s
Mu = rowMeans(NormRpm$counts)
CV2 = (apply(NormRpm$counts, 1, function(x) sd(x)/mean(x)))^2

Phi = (CV2 - 1/Mu)
Phi<-sapply(Phi, function(x) max(x,0))

#fit the mean-dispersion relationship
df<-data.frame(log.mu=log10(Mu),
                 log.phi=log10(Phi))
fit<-loess(df$log.phi~df$log.mu)

plot((df), xlab="Log Mu", ylab="Log Phi")
lines(df$log.mu,fit$fitted, col="red", lwd=1.5)

```



The scaling factors of 200 cells (divided in two groups of 100 each) are randomly chosen from the data. Moreover, 1,000 genes for which differential expression will be simulated are also selected at random among the top 50% most highly expressed genes.

```

#select 200 cells at random
set.seed(0)
cells<-sample(1:ncol(counts),size=200)

#divide the cells in two groups and store their parameters
cells.1<-cells[1:100]
cells.2<-setdiff(cells,cells.1)

S.1<-S[cells.1]
S.2<-S[cells.2]

#Randomly select genes for which differential expression will be introduced
N.genes<-1000
set.seed(1)
#10% of the genes are randomly selected in the top 10% quantile of expression
#90% of the genes randomly selected between the 50% and 90% quantile of expression
g<-c(sample(names(Mu)[Mu>quantile(Mu,0.9)])[1:(N.genes*0.1)],
      sample(names(Mu)[Mu<quantile(Mu,0.9) & Mu>quantile(Mu,0.5)])[1:(N.genes*0.9)])

```

We perform a test run of 100 simulations with the parameters estimated above and check that the simulated datasets closely resemble the real dataset in some key features (e.g., the fraction of genes with zero counts in each cell).

```
N.sim<-100

Mu.sim<-Mu
Phi.sim<-Phi
S.sim<-c(S.1,S.2)

set.seed(10)
Seeds = sample(1:1e6, size = N.sim)
TestRun <- mclapply(Seeds, function(x){
  set.seed(x)
  return(simNB_phi(Mu.sim, S.sim, Phi.sim))}, mc.cores = 1)

sim.zeros.per.cell<-lapply(TestRun, function(x){
  res<-apply(x, 2, function(y) length(which(y==0))/nrow(x))

})

sim.zeros.per.cell<-do.call(cbind, sim.zeros.per.cell)
sim.zeros.per.cell<-apply(sim.zeros.per.cell, 1, mean)

data.zeros.per.cell<-apply(counts,2, function(x) length(which(x==0))/nrow(counts))

sim.zeros.per.gene<-lapply(TestRun, function(x){
  res<-apply(x, 1, function(y) length(which(y==0))/ncol(x))
})

sim.zeros.per.gene<-do.call(cbind, sim.zeros.per.gene)
sim.zeros.per.gene<-apply(sim.zeros.per.gene, 1, mean)

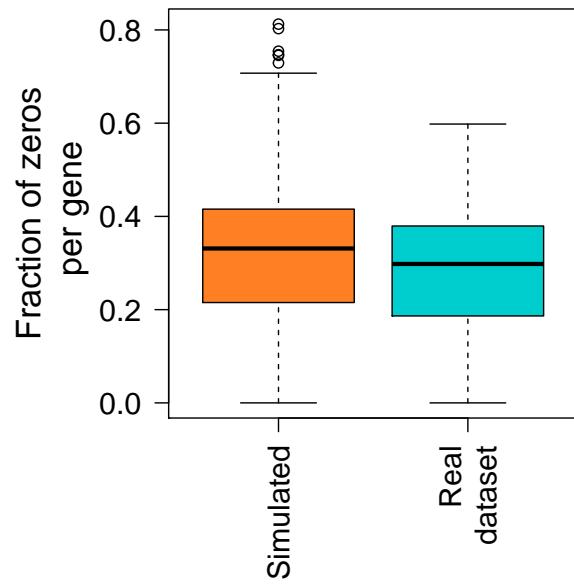
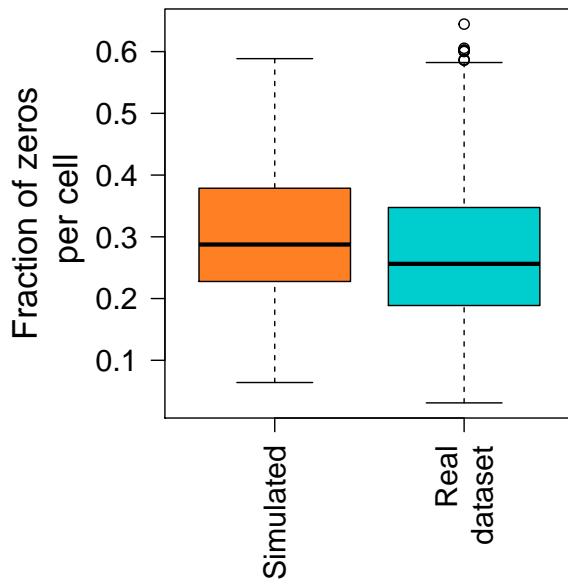
data.zeros.per.gene<-apply(counts,1, function(x) length(which(x==0))/ncol(counts))

#save simulation to file
save(file=file.path(results.path, "TestRun.RData"),
      list=c("sim.zeros.per.cell","data.zeros.per.cell",
            "sim.zeros.per.gene","data.zeros.per.gene") )

load(file.path(results.path, "TestRun.RData"))

col=c("chocolate1","cyan3")
par(cex.lab=1.5, cex.axis=1.3)
par(mfrow=c(1,2))
par(mar=c(7,7,3,3))
boxplot(sim.zeros.per.cell, data.zeros.per.cell,
        ylab="Fraction of zeros\n per cell",
        names=c("Simulated", "Real\n dataset"),col=col,
        las=2)
boxplot(sim.zeros.per.gene, data.zeros.per.gene,
        ylab="Fraction of zeros\n per gene",
```

```
names=c("Simulated", "Real\n dataset"), col=col,
las=2)
```



## Simulations

### Symmetric differential expression

The first scenario we simulate is characterised by an equal number of genes upregulated in each group.

```
fc<-5#fold change
```

```
N.sim<-100#number of simulations
```

```
#randomly choose the genes to upregulate in the two groups
set.seed(2)
g.1<-sample(g, size=(length(g)/2))
g.2<-setdiff(g,g.1)

#group 1
Mu.1<-Mu
Phi.1<-Phi
Mu.1[g.1]<-sapply(Mu.1[g.1], function(x)
  ifelse(fc*x>max(Mu), max(Mu), fc*x))
#The maximum mean expression of each gene is
#equal to the maximum average expression measured from the data
Phi.1[g.1]<-10^(predict(fit,log10(Mu.1[g.1])))
```

```
#group 2
Mu.2<-Mu
Phi.2<-Phi
Mu.2[g.2]<-sapply(Mu.2[g.2], function(x)
  ifelse(fc*x>max(Mu), max(Mu), fc*x))
Phi.2[g.2]<-10^(predict(fit,log10(Mu.2[g.2])))
```

```

set.seed(3)
Seeds = sample(1:1e6, size = N.sim)
SymmDE <- simplify2array(mclapply(Seeds, function(x)
  RunSim(seed = x, mu = list(Mu.1,Mu.2), s = list(S.1,S.2),
         phi = list(Phi.1,Phi.2), groups = "yes"), mc.cores = 1))

#save simulation to file
save(file=file.path(results.path, "symmDE.RData"), list="SymmDE")

```

## Asymmetric differential expression

We now simulate an asymmetric distribution of upregulated genes across the two groups of cells: 60%/40%, 80%/20% and 100%/0%.

```

fc<-5#fold change
N.sim<-100#number of simulations

#randomly choose the genes to upregulate in the two groups
set.seed(2)
g.1<-sample(g, size=(length(g)*0.6))
g.2<-setdiff(g,g.1)

#group 1
Mu.1<-Mu
Phi.1<-Phi
Mu.1[g.1]<-sapply(Mu.1[g.1], function(x)
  ifelse(fc*x>max(Mu), max(Mu), fc*x))
#The maximum mean expression of each gene is
#equal to the maximum average expression measured from the data
Phi.1[g.1]<-10^(predict(fit,log10(Mu.1[g.1])))

#group 2
Mu.2<-Mu
Phi.2<-Phi
Mu.2[g.2]<-sapply(Mu.2[g.2], function(x)
  ifelse(fc*x>max(Mu), max(Mu), fc*x))
Phi.2[g.2]<-10^(predict(fit,log10(Mu.2[g.2])))

set.seed(5)
Seeds = sample(1:1e6, size = N.sim)
ASymmDE60 <- simplify2array(mclapply(Seeds, function(x)
  RunSim(seed = x, mu = list(Mu.1,Mu.2), s = list(S.1,S.2),
         phi = list(Phi.1,Phi.2), groups = "yes"), mc.cores = 1))

```

```

#save simulation to file
save(file=file.path(results.path, "asymmDE60.RData"), list="ASymmDE60")

fc<-5#fold change
N.sim<-100#number of simulations

#randomly choose the genes to upregulate in the two groups
set.seed(2)
g.1<-sample(g, size=(length(g)*0.8))
g.2<-setdiff(g,g.1)

#group 1
Mu.1<-Mu
Phi.1<-Phi
Mu.1[g.1]<-sapply(Mu.1[g.1], function(x)
  ifelse(fc*x>max(Mu), max(Mu), fc*x))
#The maximum mean expression of each gene is
#equal to the maximum average expression measured from the data
Phi.1[g.1]<-10^(predict(fit,log10(Mu.1[g.1])))

#group 2
Mu.2<-Mu
Phi.2<-Phi
Mu.2[g.2]<-sapply(Mu.2[g.2], function(x)
  ifelse(fc*x>max(Mu), max(Mu), fc*x))
Phi.2[g.2]<-10^(predict(fit,log10(Mu.2[g.2])))

set.seed(6)
Seeds = sample(1:1e6, size = N.sim)
ASymmDE80 <- simplify2array(mclapply(Seeds, function(x)
  RunSim(seed = x, mu = list(Mu.1,Mu.2), s = list(S.1,S.2),
         phi = list(Phi.1,Phi.2), groups = "yes"), mc.cores = 1))

#save simulation to file
save(file=file.path(results.path, "asymmDE80.RData"), list="ASymmDE80")

fc<-5#fold change
N.sim<-100#number of simulations

#randomly choose the genes to upregulate in the two groups
set.seed(2)
g.1<-g

#group 1
Mu.1<-Mu
Phi.1<-Phi

```

```

Mu.1[g.1]<-sapply(Mu.1[g.1], function(x)
  ifelse(fc*x>max(Mu), max(Mu), fc*x))
#The maximum mean expression of each gene is
#equal to the maximum average expression measured from the data
Phi.1[g.1]<-10^(predict(fit,log10(Mu.1[g.1])))

#group 2
Mu.2<-Mu
Phi.2<-Phi

set.seed(7)
Seeds = sample(1:1e6, size = N.sim)
ASymmDE100 <- simplify2array(mclapply(Seeds, function(x)
  RunSim(seed = x, mu = list(Mu.1,Mu.2), s = list(S.1,S.2),
  phi = list(Phi.1,Phi.2), groups = "yes"), mc.cores = 1))

#save simulation to file
save(file=file.path(results.path, "asymmDE100.RData"), list="ASymmDE100")

```

## Results

We plot below the ratio of estimated and true scaling factors for each of the tested normalization methods.

```

# #load results
load(file.path(results.path, "symmDE.RData"))
load(file.path(results.path, "asymmDE60.RData"))
load(file.path(results.path, "asymmDE80.RData"))
load(file.path(results.path, "asymmDE100.RData"))

#collect and re-normalize the true scaling factors
n.cells<-length(cells)
true.size<-c(S.1,S.2)
true.size<-n.cells*(true.size/sum(true.size))

#Compute average ratio between simulated and true scaling factors
ratios<-list(SymmDE=compute.ratio(Sim=SymmDE,true.size=true.size),
  ASymmDE60=compute.ratio(Sim=ASymmDE60, true.size=true.size),
  ASymmDE80=compute.ratio(Sim=ASymmDE80, true.size=true.size),
  ASymmDE100=compute.ratio(Sim=ASymmDE100, true.size=true.size))

```

```

#plot results
colors=c(rep("lightpink3",2),
        rep("darkolivegreen3",2),
        rep("darkgoldenrod1",2),
        rep("lightskyblue1",2),
        rep("magenta1",2))
names=c("RPM,group1", "RPM,group2",
       "DESeq,group1", "DESeq,group2",
       "TMM,group1", "TMM,group2",
       "UQ,group1", "UQ,group2",
       "Scran,group1", "Scran,group2")

par(mfrow=c(2,2))
par(mar=c(9,6,3,3))
par(cex.lab=1.5, cex.axis=1.2)
boxplot(ratios$SymmDE, las=2, col=colors,
        ylab="Scaling factor\n (Estimated/True)", names=names,
        main="Symmetric DE")
abline(h=1, lwd=2, col="red")

boxplot(ratios$ASymmDE60, las=2, col=colors,
        ylab="Scaling factor\n (Estimated/True)", names=names,
        main="Asymmetric DE (60/40)")
abline(h=1, lwd=2, col="red")

boxplot(ratios$ASymmDE80, las=2, col=colors,
        ylab="Scaling factor\n (Estimated/True)", names=names,
        main="Asymmetric DE (80/20)")
abline(h=1, lwd=2, col="red")

boxplot(ratios$ASymmDE100, las=2, col=colors,
        ylab="Scaling factor\n (Estimated/True)", names=names,
        main="Fully asymmetric DE")
abline(h=1, lwd=2, col="red")

```

